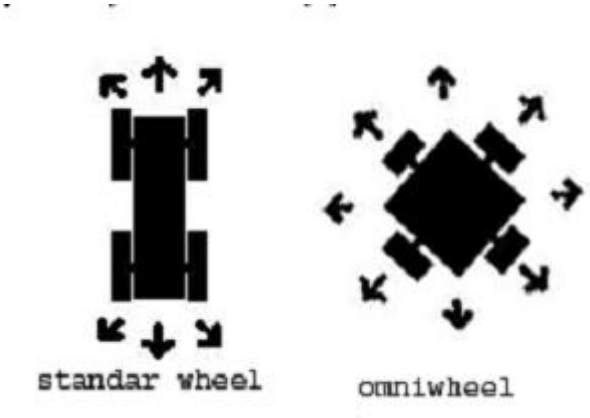
BAB II

LANDASAN TEORI

2.1. *Omni-Directional Robot*

Omni-directional robot adalah robot dengan sistem pergerakan yang secara langsung dapat bergerak kesegala arah dengan konfigurasi apapun. Pada umumnya robot di desain dengan pergerakan yang sudah direncanakan terlebih dahulu. Pada sistem pergerakkan konvensional, pergerakkan tidak mampu di kontrol pada setiap tingkat kebebasan dalam bergerak secara *independent*, sehingga hanya mampu bergerak ke beberapa arah yang sudah ditentukkan sebelumnya. Ini disebut kendala *non-holomic* yaitu pencegahan roda kemudi dari selip, meskipun pada umumnya mampu menjangkau setiap lokasi dan orientasi dalam ruang 2 dimensi, namun memerlukan manuver dan perencanaan jalan yang rumit dan kompleks. (Doroftei, 2007)

Keunggulan robot *omni* ini adalah pada roda yang berupa *omni directional poly roller wheel*. Pada Gambar 2.1 terlihat sebuah robot dengan *omniwheel* mampu melakukan gerakan yang kompleks untuk mencapai posisi tertentu. Dengan sistem pergerakkan ini maka robot akan memiliki 2 derajat kebebasan karena dapat bergerak pada aksis x ataupun y. Pada umumnya desain robot *omni* terdapat 2 jenis yaitu robot *omni* dengan 3 roda dan 4 roda. Pengaturan posisi *omniwheel* mempengaruhi pergerakan robot secara signifikan, jika dengan *omniwheel* standar semakin jauh jarak roda depan dengan roda belakang maka semakin cepat untuk memutar posisi robot (Syam, 2011)



Gambar 2.1 Sistem Pergerakan Konvensional dan *Omni-Directional*

(Syam, 2011)

Omniwheel terdiri dari roda inti besar dan sepanjang *peripheral* ada terdapat banyak roda kecil tambahan yang mempunyai poros tegak lurus pada roda inti. *Omniwheel* merupakan roda *mecanum* dengan cakram kecil di sekitar lingkar yang tegak lurus terhadap arah bergulir. Efeknya adalah bahwa roda akan berputar dengan kekuatan penuh, dan akan bergeser dengan sangat mudah. *Omniwheel* ini sering digunakan dalam sistem penggerak *holonomic*. Sistem pergerakkan ini sering digunakan dalam perlombaan seperti RoboCup, robot banyak menggunakan *omniwheel* ini karena memiliki kemampuan untuk bergerak ke segala arah. Roda *omni* sering digunakan sebagai kastor bertenaga untuk robot berkendara diferensial untuk membuat berputar lebih cepat. (Syam, 2011). Berikut beberapa jenis *omniwheel* seperti pada Gambar 2.2.

Gambar 2.2 Jenis *Omniwheel* (Syam, 2011)

2.2. Robotino

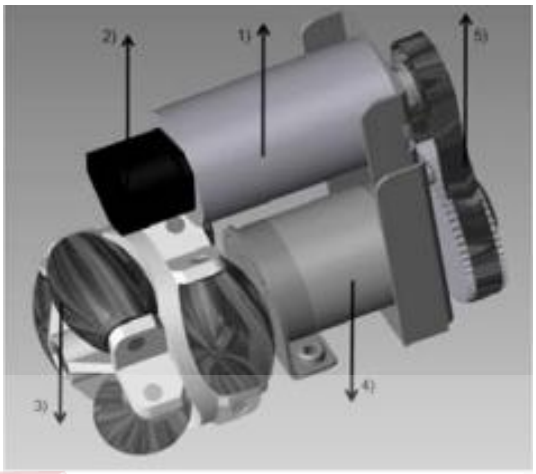
Robotino adalah robot buatan Festo Didactic yang digunakan untuk edukasi dan penelitian serta kompetisi robot. Robotino memiliki fitur sistem gerak menggunakan *omni-directional drive, bumps sensors, infrared distance sensors,* dan *usb webcam.* Robotino didesain modular, sehingga dapat dengan mudah ditambahkan berbagai akesesoris pelengkap, seperti *sensor laser scanner, gyroscope,* dan *postioning system northstar* dalam ruangan. (ROS, 2010). Gambar Robotino dapat dilihat pada Gambar 2.3.

Gambar 2.3 Robotino (ROS, 2010)

Robotino dapat bergerak maju, mundur dan menyamping ke segala arah, serta berputar di tempat, dengan menggunakan tiga roda. Robot ini dapat diintegrasikan dan digunakan sebagai pilihan teknologi, misalnya untuk teknologi penggerak listrik, sensor, teknologi kontrol, pengolahan citra dan teknik pemrograman. (Karras, 2009). Robotino memiliki spesifikasi *hardware* sebagai berikut :

1. Diameter: 370 mm
2. Tinggi: 210 mm
3. Berat secara keseluruhan: 11 kg
4. Karet penjaga strip (*bumper*) disekeliling robot untuk perlindungan sensor tabrakan
5. 9 sensor jarak infra merah
6. Sensor analog induktif
7. 2 sensor optik
8. *Webcam* dengan antarmuka USB
9. Wireless LAN
10. Dapat diperluas dengan menggunakan dua busi 20-pin
11. 2 *power supply* 12V akumulator gel
12. *Ethernet*, VGA dan USB untuk koneksi langsung ke *monitor* dan *keyboard*
13. *Driver unit*, terdapat 3 *unit* yang terdiri dari komponen DC *motor, integrated planetary gear, omni-directional wheels, toothed belt with gear wheels,* dan incremental encoder. Kecepatan motor akan dikontrol melalui pengontrol PID yang diimplementasikan pada *atmel microprocessor* sebagai pusat

pengolah data pada robotino. Gambar *driver unit* dapat dilihat pada Gambar 2.4.



Gambar 2.4 *Driver unit* (Karras, 2009)

2.3. *Webcam*

Web camera atau yang biasa dikenal dengan *webcam*, adalah kamera yang gambarnya bisa diakses menggunakan *world wide web* (www), program *instant messaging*, atau aplikasi komunikasi dengan tampilan *video* pada PC. *Webcam* juga digambarkan sebagai kamera *video* digital yang sengaja didesain sebagai kamera dengan resolusi rendah. *Webcam* dapat digunakan untuk sistem keamanan. Pada beberapa *webcam*, ada yang dilengkapi dengan *software* yang mampu mendeteksi pergerakan dan suara. Dengan *software* tersebut, memungkinkan PC yang terhubung ke kamera untuk mengamati pergerakan dan suara, serta merekamnya ketika terdeteksi. Hasil rekaman ini bisa disimpan pada komputer, e-mail atau di *upload* ke internet. (Wibowo, 2010)

Webcam sangat bermanfaat dalam bidang telekomunikasi, bidang keamanan dan bidang industri. Sebagai contoh *webcam* digunakan untuk *video call chatting*, *surveillience camera*, dan sebagai *video conference* oleh beberapa *user*. Namun seiring perkembangan zaman, *webcam* sekarang dimanfaatkan sebagai sensor pada robot yang digunakan sebagai mata robot yang akan menangkap gambar dan diolah dalam citra digital untuk menghasilkan gerakan yang diinginkan.

Gambar 2.5 *Webcam*

Sebuah *web camera* yang sederhana terdiri dari sebuah lensa standar, dipasang di sebuah papan sirkuit untuk menangkap sinyal gambar; *casing (cover)*, termasuk *casing* depan dan *casing* samping untuk menutupi lensa standar dan memiliki sebuah lubang lensa di casing depan yang berguna untuk memasukkan gambar; kabel *support*, yang dibuat dari bahan yang fleksibel, salah satu ujungnya dihubungkan dengan papan sirkuit dan ujung satu lagi memiliki *connector*, kabel ini dikontrol untuk menyesuaikan ketinggian, arah dan sudut pandang *web camera*. Sebuah *web camera* biasanya dilengkapi dengan *software*, *software* ini mengambil gambar-gambar dari kamera digital secara terus menerus ataupun dalam interval waktu tertentu dan menyiarkannya melalui koneksi internet. Ada beberapa metode penyiaran, metode yang paling umum adalah *hardware* mengubah gambar ke dalam

bentuk file JPG dan menguploadnya ke *web server* menggunakan *File Transfer Protocol* (FTP).

Frame rate mengindikasikan jumlah gambar sebuah *software* dapat ambil dan transfer dalam satu detik. Untuk *streaming video*, dibutuhkan minimal 15 *frame per second* (fps) atau idealnya 30 fps. Untuk mendapatkan *frame rate* yang tinggi, dibutuhkan koneksi internet yang tinggi kecepatannya. Sebuah *web camera* tidak harus selalu terhubung dengan komputer, ada *web camera* yang *memiliki software webcam* dan *web server bulit-in*, sehingga yang diperlukan hanyalah koneksi internet. *Web camera* seperti ini dinamakan *"network camera"*. Kita juga bisa menghindari penggunaan kabel dengan menggunakan hubungan radio, koneksi *Ethernet* ataupun *WiFi*.

2.4. *Citra Digital*

Citra digital adalah matriks dua dimensi yang dapat ditampilkan pada layar monitor komputer sebagai himpunan berhingga (diskrit) nilai digital yang disebut piksel (*picture elements*). Piksel adalah elemen citra yang memiliki nilai yang menunjukkan intensitas warna. Berdasarkan cara penyimpanan atau pembentukannya, citra digital dapat dibagi menjadi dua jenis.

Jenis pertama adalah citra digital yang dibentuk oleh kumpulan piksel dalam array dua dimensi. Citra jenis ini disebut citra *bitmap* atau citra *raster*. Jenis citra yang kedua adalah citra yang dibentuk oleh fungsi-fungsi geometri dan matematika. Jenis citra ini disebut grafik vektor.

Citra digital (diskrit) dihasilkan dari citra analog (*continue*) melalui digitalisasi. Digitalisasi citra analog terdiri s*ampling* dan *quantitazion Sampling*

adalah pembagian citra ke dalam elemen-elemen diskrit (piksel), sedangkan *quantitazion* adalah pemberian nilai intensitas warna pada setiap piksel dengan nilai yang berupa bilangan bulat. (Awcock, 1996)

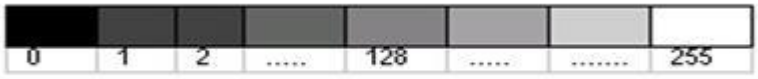
2.5. Pengolahan Citra Digital

Pengolahan citra merupakan teknik manipulasi citra secara digital yang khususnya menggunakan komputer, menjadi citra lain yang sesuai untuk digunakan dalam aplikasi tertentu. Agar mudah diinterpretasi oleh manusia atau komputer, pengolahan citra harus dilakukan dengan berbagai macam metode untuk mencapai citra sesuai yang diinginkan. Operasi pengolahan citra digital umumnya dilakukan dengan tujuan memperbaiki kualitas suatu gambar sehingga dapat dengan mudah diinterpretasikan oleh mata manusia dan untuk mengolah informasi yang ada pada suatu gambar untuk kebutuhan identifikasi objek secara otomatis (Murinto, Ariwibowo, & Syazali, 2009).

Operasi-operasi pada pengolahan citra digital secara umum terbagi menjadi beberapa macam kuaifikasi antara lain: perbaikan kualitas citra *(image enhancement),* restorasi citra *(image restoration),* pemampatan citra *(image compression),* segmentasi citra *(image segmentation),* pengorakan citra *(image analysis),* rekonstruksi citra *(image recronstruction).* Dalam penilitian kali ini pengolahan citra digital yang digunakan adalah citra *grayscale,* dan histogram *ekualisasi.*

2.5.1. Citra *Grayscale*

Citra *grayscale* merupakan citra digital yang hanya memiliki satu nilai pada setiap pikselnya (8 bit). Citra yang ditampilkan dari citra jenis ini terdiri atas warna abu-abu, bervariasi pada warna hitam pada bagian intensitas terlemah (0) dan warna putih pada bagian intensitas terkuat (255). (Fatta, 2007)



Gambar 2.6 Derajat intensitas nilai keabuan (*grayscale*).

Citra *grayscale* merupakan citra yang didapat dari gambar berwarna yang memiliki tiga nilai di tiap pikselnya (24 bit) atau RGB dengan cara mengambil nilai rata-rata RGB pada suatu piksel.

$$f_{Gray} = \frac{f_{Red} + f_{Green} + f_{Blue}}{3} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.1)$$

Untuk mendapatkan citra *grayscale* bisa juga dengan menggunakan cara perskalaan, yaitu dengan mengalikan nilai R, G dan B dengan 0.299R, 0.587G, 0.114B lalu menjumlahkan hasil dari perkalian tadi.

2.5.2. Histogram Ekualisasi

Histogram citra adalah grafik yang menggambarkan penyebaran nilai- nilai intensitas piksel dari suatu citra. Dari sebuah histogram dapat diketahui frekuensi kemunculan nisbi (relatif) citra tersebut. Histogram merupakan gambaran variasi nilai intensitas piksel secara menyeluruh atau menunjukkan distribusi frekuensi kemunculan nilai piksel data citra. Suatu citra kadang-kadang mempunyai

histogram yang sangat sempit. Bila histogramnya mengumpul di sebelah kiri, citra akan kelihatan gelap, sebaliknya, citra akan sangat terang jika histogramnya mengumpul di sebelah kanan. Histogram juga dapat menunjukkan banyak hal tentang kecerahan (brightness) dan kontras dari sebuah citra. Secara matematis histogram citra dapat dihitung dengan persamaan 2.2 :

$$h_i = \frac{n_i}{n}, i = 0,1,2,\ldots,L-1 \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.2)$$

dengan n = jumlah seluruh piksel didalam citra dan n_i = jumlah piksel yang memiliki derajat keabuan i.

Distribusi h_i, atau n_i, dapat menyediakan informasi tentang kemunculan citra. Pengetahuan praktis untuk memahami histogram citra dibutuhkan untuk melihat perubahan-perubahan pada citra setelah dilakukan operasi tertentu. Beberapa pengetahuan praktis yang biasa digunakan dalam melihat histogram citra adalah sebagai berikut:

1. Histogram citra yang terdistribusi merata pada seluruh tingkat keabuan memiliki kontras yang baik.
2. Histogram citra yang mengumpul pada daerah gelap memiliki citra redup.
3. Histogram citra yang mengumpul pada daerah terang atau terkonsentrasi pada intensitas citra yang tinggi menampilkan citra yang terang.

Disamping itu dalam upaya menampakkan informasi sebanyak mungkin pada citra maka histogram dibuat semerata mungkin yang disebut dengan penyamaan histogram (histogram equalization). Tujuan ekualisasi histogram adalah untuk memperoleh citra keluaran dengan histogram yang seragam, sehingga setiap derajat keabuan memiliki jumlah piksel yang relatif sama. (Munir, 2004). Yang dimaksud dengan perataan histogram adalah mengubah derajat keabuan suatu piksel (r)

dengan derajat keabuan yang baru (s) dengan suatu fungsi transformasi T, yang dalam hal ini s = T(r). Dua sifat yang dipertahankan pada tranformasi ini:

1. Nilai s merupakan pemetaan 1 ke 1 dari r. Ini untuk menjamin representasi intensitas yang tetap. Ini berarti r dapat diperoleh kembali dari r dengan transformasi invers: $r = T^{-1}(s)$, $0 \leq s \leq 1$.
2. Untuk $0 \leq ri \leq 1$, maka $0 \leq T(r) \leq 1$. Ini untuk menjamin pemetaan T konsisten pada rentang nilai yang diperbolehkan.

2.6. *Principal Component Analysis (PCA)*

Principal component analysis atau *Karhunen Loeve Transform* merupakan salah satu metode pengenalan wajah yang memiliki dasar pada perhitungan statistik dan matematika. PCA termasuk dalam algoritma pengenalan wajah yang termasuk pada golongan *appearance based*. (Mohammed, 2004). Metode ini mempunyai komputasi yang sederhana dan cepat dibandingkan dengan penggunaan metode yang memerlukan banyak pembelajaran seperti jaringan saraf tiruan.

PCA adalah teknik statistik untuk menyederhanakan kumpulan data yang dimensi besar menjadi dimensi yang lebih rendah. Metode ini merupakan metode yang bekerja pada wilayah linier, maka aplikasi yang memiliki model linier dapat menggunakan metode ini, seperti pengolahan sinyal, pengolahan citra, dan lain-lain. Konsep penggunaan PCA meliputi perhitungan nilai-nilai simpangan baku, matriks kovarian, nilai karakteristik (*eigenvalue*), dan vector karakteristik (*eigenvector*). PCA dapat menggunakan metode kovariansi atau korelasi. Jika diperlukan, data dapat distandarisasi terlebih dahulu sehingga mendekati sebaran

normal baku. Dalam hal ini digunakan metode kovariansi dengan algoritma sebagai berikut :

1. Cari rata-rata dari X, cara menghitung rata-rata :

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.3)$$

2. Lakukan normalisasi data agar PCA dapat bekerja dengan benar, setiap data asli akan dikurangkan dengan rata-rata, selisih rata-rata :

$$\varphi = X - \bar{X} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.4)$$

3. Setelah didapat normalisasi data, selanjutnya adalah melakukan perhitungan kovarian dari matriks X = $[\varphi_1 \ \varphi_2 \ \varphi_3 \ \cdots \ \varphi_M]$, dengan cara :

$$C = \frac{1}{M}\sum_{i=1}^{M} \varphi_i \ {\varphi_i}^T \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.5)$$

4. Setelah didapat nilai kovariannya langkah selanjutnya adalah menentukan nilai eigen dan vektor eigen dari matriks kovarian C : $\lambda_1 > \lambda_2 > \lambda_3 > \ldots > \lambda_M$ dan C : $\mu_1 \ \mu_2 \ \mu_3 \ldots \mu_M$.
5. Mengurutkan vektor eigen μ yang bersesuaian dengan nilai eigen λ yang didapat dari matriks kovarian, mulai dari yang terbesar ke yang terkecil. Hal ini dimaksudkan agar didapat vektor eigen yang tersusun berdasarkan pengaruhnya.

2.7. Algoritma *Eigen Image*

2.7.1. Nilai Eigen suatu matriks

Jika diketahui matriks A berukuran n x n, $\bar{x}$ vektor tak nol berukuran n x 1, $\bar{x} \in R^n$. Karena A berukuran n x n, maka A$\bar{x}$ akan berupa vektor yang berukuran n x 1 juga. Bila terdapat skalar λ, $\lambda \in Riil$, sedemikian hingga

$$A\bar{x} = \lambda\bar{x}, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.6)$$

Semua nilai λ yang memenuhi persamaan tersebut sehingga ada nilai $\bar{x}$ yang nyata (bukan vektor $\bar{0}$ saja) disebut nilai eigen (karakteristik). Untuk menemukan nilai dari λ, dari persamaan A$\bar{x}$ = λ$\bar{x}$ sebelumnya dirubah dahulu menjadi persamaan (A – λI) $\bar{x}$ = $\bar{0}$ = (λI - A) $\bar{x}$. Agar persamaan tersebut memiliki penyelesaian, maka dapat ditentukan melalui nilai det (A – λI) yaitu det (A – λI) = det (λI - A) = 0, persamaan tersebut bisa juga disebut dengan persamaan karakteristik. Banyaknya nilai eigen maksimal adalah n buah.

Dari nilai eigen yang telah diperoleh tersebut dapat ditentukan ruang solusi untuk $\bar{x}$ dengan memasukkan nilai eigen yang diperoleh kedalam persamaan (A – λI) $\bar{x}$ = 0. Ruang solusi yang diperoleh dengan cara demikian ini disebut juga dengan ruang eigen. Dari ruang eigen yang bersesuaian dengan nilai eigen tertentu tersebut dapat dicari minimal sebuah ruang eigen yang saling bebas linier.

Bagian terpenting dalam algortima eigen *image* adalah eigen value dan eigen vector. Eigen value dan eigen vector dari suatu matriks bujur sangkar selalu berkorespondensi (saling bersesuaian) satu sama lain. Jika eigen value sudah ditemukan maka pasti eigen vector juga sudah ditemukan. Berikut contoh untuk operasi nilai eigen :

Misal terdapat matriks A = $\begin{bmatrix} 3 & 2 \\ -1 & 0 \end{bmatrix}$ Langkah untuk mencari nilai eigen adalah:

$(\lambda I - A) = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 3 & 2 \\ -1 & 0 \end{bmatrix}$

$= \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 3 & 2 \\ -1 & 0 \end{bmatrix}$

$= \begin{bmatrix} \lambda - 3 & -2 \\ 1 & \lambda \end{bmatrix}$

Det $(\lambda I - A) = \det \begin{bmatrix} \lambda - 3 & -2 \\ 1 & \lambda \end{bmatrix}$

$= \lambda^2 - 3\lambda + 2 = 0$

$(\lambda - 2)(\lambda - 1) = 0 >> \lambda = 2$ dan $\lambda = 1$

Mencari vektor eigen :

untuk $\lambda = 2 >> \begin{bmatrix} \lambda - 3 & -2 \\ 1 & \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 2 - 3 & -2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\begin{bmatrix} -1 & -2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$-x_1 - 2x_2 = 0$

$x_1 + 2x_2 = 0$

Ambil satu persamaan, misal $x_1 + 2x_2$

$x_1 = -2x_2$

Didapat $x_1 = -2 \; dan \; x_2 = 1$

Maka eigen vektor untuk $\lambda = 2$ adalah $\begin{bmatrix} -2 \\ 1 \end{bmatrix}$, hal sama dilakukan juga untuk mencari eigen vektor jika $\lambda = 1$. (Rizkananda, 2010)

2.7.2. Eigen *Image*

Eigen *image* adalah kumpulan dari eigen vektor yang digunakan untuk masalah computer vision pada pengenalan gambar, seperti pengenalan wajah manusia. Untuk menghasilkan eigen *image*, sekumpulan besar citra digital dari data gambar diambil pada kondisi pencahayaan yang sama dan kemudian dinormalisasi, lalu diolah pada resolusi yang sama (misalnya m x n), dan kemudian diperlakukan sebagai vektor dimensi mn dimana komponennya diambil dari nilai dari pikselnya. (Sarwono, 2011)

2.7.3. Algoritma Eigen *Image* dengan PCA

Eigen *image* adalah salah satu pengenalan gambar yang didasarkan pada principal component analysis (PCA) yang dikembangkan di MIT. Gambar pelatihan direpresentasikan dalam sebuah vektor flat (gabungan vektor) dan digabung bersama-sama menjadi sebuah matriks tunggal. Eigen *image* dari masing-masing citra kemudian di ekstrak dan disimpan dalam database. Gambar yang akan dicobakan untuk pengenalan juga akan didefinisikan nilai eigen *image*-nya dibandingkan dengan eigen *image* dari gambar pelatihan atau data dari database. Berikut adalah algoritma perhitungan eigen *image* dengan menggunakan PCA :

1. Ambil data dari gambar inputan (gambar pelatihan).
2. Ubah data gambar menjadi sebuah vektor kolom.
3. Kumpulkan semua data yang akan diproses dalam sebuah matriks X (berukuran M x N). Dimana M = jumlah gambar input dan N = baris x kolom.
4. Cari rata-rata dari X, cara menghitung rata-rata :

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.7)$$

5. Lakukan normalisasi data agar PCA dapat bekerja dengan benar, setiap data asli akan dikurangkan dengan rata-rata, selisih rata-rata :

$$\varphi = X - \bar{X} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.8)$$

6. Setelah didapat normalisasi data, selanjutnya adalah melakukan perhitungan kovarian dari matriks X = [$\varphi_1\ \varphi_2\ \varphi_3\ \cdots\ \varphi_M$], dengan rumus :

$$C = \frac{1}{M}\sum_{i=1}^{M} \varphi_i\ {\varphi_i}^T = XX^T\ (matriks\ N^2\ x\ N^2) \quad \cdots\cdots\cdots\cdots\cdots (2.9)$$

7. Menggunakan metode PCA pada perhitungan matriks kovarian C untuk mencari eigen vektor dan *eigen value*. Matriks kovarian C ordo N^2 x N^2 tidak efektif karena data yang diproses terlalu besar. Oleh karena itu dengan memanfaatkan sifat perkalian matriks, pada metode PCA digunakan persamaan 2.9 :

$$L = X^T X \quad (matriks\ M\ x\ M) \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.10)$$

8. Setelah didapat nilai kovariannya langkah selanjutnya adalah menentukan nilai eigen dan vektor eigen dari matriks kovarian C : $\lambda_1 > \lambda_2 > \lambda_3 > \ldots > \lambda_M$ dan C : $\mu_1\ \mu_2\ \mu_3 \ldots \mu_M$.
9. Mengurutkan vektor eigen μ yang bersesuaian dengan nilai eigen λ yang didapat dari matriks kovarian, mulai dari yang terbesar ke yang terkecil. Hal ini dimaksudkan agar didapat vektor eigen yang tersusun berdasarkan pengaruhnya. Vektor eigen yang tidak terlalu berpengaruh (memiliki nilai

kecil) bisa kita hilangkan, walaupun hal ini akan sedikit menghilangkan informasi yang ada pada data tersebut, namun hal ini akan bisa mengurangi dimensi data menjadi lebih kecil dan bisa mempercepat perhitungan. Jadi kita hanya akan menggunakan beberapa vektor eigen yang paling berpengaruh, sesuai urutannya (λ_K).

10. Langkah selanjutnya adalah untuk mendapatkan nilai data baru dengan mengalikan matriks dari vektor eigen dengan data normalisasi φ atau bisa juga disebut dengan *eigenface.*

$$Eigenface = \sum_{i=1}^{M} \mu_i \ \varphi_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.11)$$

11. Setelah didapat nilai *eigenface* lalu langkah selanjutnya adalah menghitung *eigen space* atau bisa juga disebut *weight*, cara menghitung *weight* :

$$weight = \varphi^T * eigenface \ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.12)$$

Setelah mencari nilai eigen *image* pada proses pelatihan, maka pada proses selanjutnya adalah melakukan pengenalan wajah. Pada metode PCA, langkah yang dilakukan untuk pengenalan wajah :

1. Normalisasi data, dengan cara mengurangkan citra yang didapat (Γ) dengan rata-rata.

$$\varphi_{Test} = \Gamma - \bar{X} \ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.13)$$

2. Melakukan proyeksi ke eigen space. Citra uji ditransformasikan ke komponen eigen *image*. Hasil bobot disimpan ke dalam vektor bobot Ω_{new}^T.

$$\omega_k = \mu_k^T(\Gamma_{new} - \bar{X}) \quad k = 1,2, \dots, M' \ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.14)$$

$$\Omega_{new}^{T} = [\omega_1\ \omega_2 \dots\ \omega_k] \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.15)$$

3. Euclidean distance antara 2 vektor bobot d ($\Omega_{new}, \Omega_{train}$) digunakan untuk mengukur kemiripan antara dua buah gambar i dan j. Berikut formula dari perhitungan euclidean distance. (Pentland, 1991)

$$d\left(\Omega_i, \Omega_j\right) = \sqrt{\sum_{r=1}^{n} (\Omega_j - \Omega_j)^2} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.16)$$

2.8. Matriks

Notasi suatu matriks berukuran n x m ditulis dengan $A_{n\,x\,m}$. Huruf n menyatakan jumlah baris, dan huruf m menyatakan jumlah kolom. Suatu matriks tersusun atas elemen yang dinyatakan dengan huruf kecil lalu diikuti dengan angka-angka indeks, misalnya a_{ij}. Indeks *i* menunjukkan posisi baris ke-*i* dan indeks *j* menunjukkan posisi kolom ke-*j*. (Suparno, 2011)

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \dots & a_{1j} \\ a_{21} & a_{22} & a_{23} \dots & a_{2j} \\ \vdots & \vdots & \vdots \dots & \vdots \\ a_{i1} & a_{i2} & a_{i3} \dots & a_{ij} \end{bmatrix} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.17)$$

Misal elemen-elemen matriks A_{4x4} diisi dengan suatu nilai, maka akan terlihat seperti berikut :

$$A = \begin{bmatrix} 5 & 2 & 4 & 5 \\ 7 & 2 & 1 & 6 \\ 1 & 3 & 4 & 8 \\ 1 & 2 & 2 & 4 \end{bmatrix} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.18)$$

Dari matriks A maka akan dapat dinyatakan bahwa elemen dari $A_{11} = 5, A_{12} = 2, A_{13} = 4, A_{14} = 5, A_{21} = 7, A_{22} = 2, A_{23} = 1, A_{24} = 6,$ dan seterusnya sampai pada $A_{44} = 4$.

2.9. Sistem Persamaan Linier

Sistem persamaan linear dengan metode iterasi digunakan untuk mencari nilai eigen vektor seperti pada persamaan 2.10. Eigen vektor didapat dengan mencari eigen value seperti persamaan det (A – λI) = 0. Berikut sistem persamaan linear yang terdiri dari n-persamaan dengan variabel $x_1, x_2, \dots, x_n$ dinyatakan dengan persamaan :

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ \vdots \quad + \quad \vdots \quad + \dots + \quad \vdots \quad &= \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n \end{aligned} \quad \cdots\cdots\cdots\cdots \quad (2.19)$$

Persamaan (2.19) dapat diekspresikan dengan bentuk perkalian matriks. Sistem persamaan linear dapat diselesaikan dengan metode langsung atau metode iterasi. Kedua metode tersebut mempunyai kelemahan dan keunggulan. Metode yang dipilih akan menentukan keakuratan penyelesaian sistem tersebut. Dalam kasus tertentu, yaitu sistem yang besar, metode iterasi lebih cocok digunakan. Dalam menetukan penyelesaian sistem persamaan linear, metode iterasi menggunakan algoritma secara rekursif. Algoritma tersebut dilakukan sampai diperoleh suatu nilai yang konvergen dengan toleransi yang diberikan. Ada dua metode iterasi yang sering digunakan, yaitu metode jacobi dan metode gauss seidel. (Sarwono, 2011).

2.9.1. Metode Jacobi

Persamaan ke-i dalam Persamaan 2.19 dinyatakan sebagai berikut :

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ii}x_i + \cdots + a_{in}x_n = b_i\,,\ \ i = \ 1,2,3,\ldots,n \cdots\cdots\cdots (2.20)$$

Persamaan (2.19) dapat diekspresikan sebagai :

$$a_{ii}x_i + \sum_{j=1\,,j\neq i}^{n} a_{ij}x_j = b_i \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2.21)$$

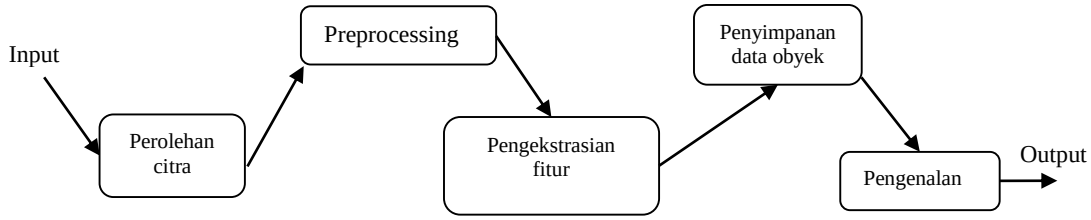
Dengan demikian, algoritma metode jacobi diekspresikan menjadi :

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1\,,j\neq i}^{n} a_{ij}x_i^{(k)}\right], dimana\ k = 0,1,2,\ldots,n \cdots\cdots (2.22)$$

Untuk menyelesaikan sistem persamaan linear dengan metode jacobi diperlukan suatu nilai pendekatan awal yaitu $x_i^{(0)}$. Nilai $x_i^{(0)}$ biasanya tidak diketahui dan dipilih $x_i^{(0)} = 0$. (Suparno, 2011). Jadi $x_i^{(1)}$ dari persamaan 2.22 akan menjadi $x_i^{(1)} = \frac{b_i}{a_{ii}}$.

2.10. *Computer Vision*

Computer Vision adalah pencitraan komputer dimana aplikasi tidak melibatkan manusia dalam proses pengulangan visual. Dengan kata lain, gambar yang diperiksa dan di olah oleh komputer. Meskipun orang yang terlibat dalam pengembangan sistem aplikasi, akhirnya membutuhkan komputer untuk mengambil informasi visual secara langsung (Umbaugh, 1998).



Gambar 2.7 Sistem visi komputer

Computer vision merupakan sebuah proses otomatis yang menintegrasikan sejumlah besar proses persepsi visual, seperti pengolahan citra, klasifikasi citra, pengenalan citra dan akusisi citra. *Computer vision* didefinisikan sebagai salah satu cabang ilmu pengetahuan yang mempelajari bagaimana komputer dapat mengenali objek yang diamati atau diobservasi. Cabang ilmu ini bersama kecerdasan buatan (*artificial intelligence*) akan mampu menghasilkan sistem kecerdasan visual (*visual intelligence system*). (Munir, 2004)

Vision = Geometri + Measurement + Interpretatio …… (2.23)

Proses-proses dalam *computer vision* dapat dibagi menjadi tiga aktivitas:

1. Memperoleh atau mengakuisisi citra digital.
2. Melakukan teknik komputasi untuk memperoses atau memodifikasi data citra.

Menganalisis dan menginterpretasi citra dan menggunakan hasil pemrosesan untuk tujuan tertentu, misalnya memandu robot, mengontrol peralatan, memantau proses manufaktur, dan lain-lain.

2.11. *Sistem Deteksi Wajah dengan Haar-Like*

Penelitian mengenai deteksi dan pengenalan wajah pada teknologi computer vision telah banyak dilakukan, salah satunya adalah menggunakan Haar

like feature yang dikenal sebagai Haar Cascade Classifier. Haar-like merupakan rectangular (persegi) features, yang memberikan indikasi secara spesifik pada sebuah gambar. Ide dari Haar-like features adalah untuk mengenali obyek yang berdasarkan nilai sederhana dari fitur tetapi bukan merupakan nilai piksel dari obyek tersebut. Metode ini memiliki kelebihan yaitu komputasinya sangat cepat, karena hanya bergantung pada jumlah piksel dalam persegi bukan setiap piksel dari obyek. Metode ini merupakan metode yang menggunakan statistical model (classifier). Pendekatan untuk mendeteksi obyek dalam gambar dengan menggabungkan empat konsep utama, antara lain:

1. *Training* data.

Metode ini memerlukan 2 tipe gambar objek dalam proses *training* yang dilakukan, yaitu :

1. *Positive samples*, Berisi gambar obyek yang ingin di deteksi, apabila ingin mendeteksi mata maka *positive samples* ini berisi gambar wajah, begitu juga obyek lain yang ingin dikenali.

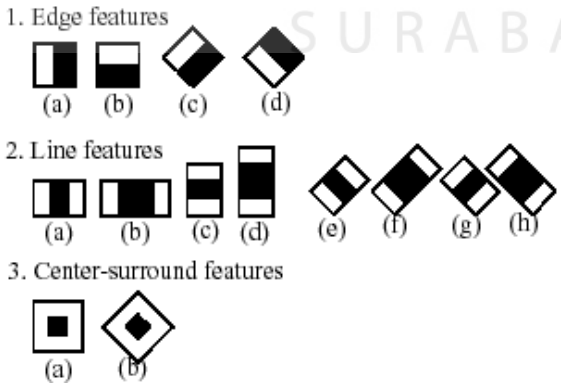
2. *Negative samples*, Berisi gambar obyek selain obyek yang ingin dikenali, umumnya berupa gambar *background* (tembok, pemandangan, lantai, dan gambar lainnya). Resolusi untuk sampel negatif disarankan untuk memiliki resolusi yang sama dengan resolusi kamera.

Training dari Haar menggunakan dua tipe sampel diatas. Informasi dari hasil *training* ini lalu dikonversi menjadi sebuah parameter model statistik.

2. Haar Feature

Algoritma Haar menggunakan metode statistical dalam melakukan pendeteksian wajah. Metode ini menggunakan sample *haarlike faetures*. Classifier ini menggunakan gambar berukuran tetap (umumnya berukuran 24x24). Cara kerja dari haar dalam mendeteksi wajah adalah dengan menggunakan teknik *sliding window* berukuran 24x24 pada keseluruhan gambar dan mencari apakah terdapat bagian dari gambar yang berbentuk seperti wajah atau tidak. *Haar* juga memiliki kemampuan untuk melakukan *scaling* sehingga dapat mendeteksi adanya wajah yang berukuran lebih besar ataupun lebih kecil dari gambar pada *classifier*. Tiap *feature* dari *haar-like feature* didefinisikan pada bentuk dari *feature*, diantaranya

koordinat dari *feature* dan juga ukuran dari *feature* tersebut. Selanjutnya kombinasi-kombinasi kotak yang digunakan untuk melakukan pendeteksian obyek visual yang lebih baik. Setiap *Haar-like features* terdiri dari gabungan kotak-kotak hitam dan putih.



Gambar 2.8 Macam-macam variasi *feature* pada *Haar*

3 tipe kotak(rectangular) feature :

1. Tipe *two-rectangle feature* (horisontal/vertikal)

2. Tipe three-rectangle feature

3. Tipe four-rectangle feature

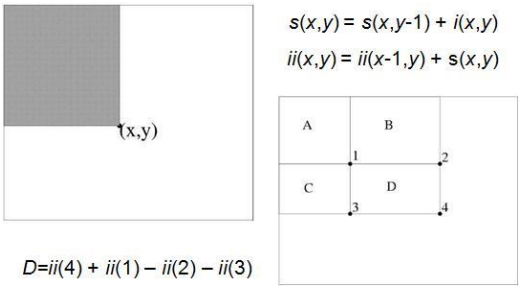
Adanya fitur Haar ditentukan dengan cara mengurangi rata-rata piksel pada daerah gelap dari rata-rata piksel pada daerah terang. Jika nilai perbedaannya itu diatas *treshold*, maka dapat dikatakan bahwa fitur tersebut ada. Nilai dari *Haar-like feature* adalah perbedaan antara jumlah nilai-nilai piksel *gray level* dalam daerah kotak hitam dan daerah kotak putih:

F(x) = SumBlack rectangle * SumWhite rectangle …… (2.24)

dimana untuk kotak pada Haar-like feature dapat dihitung secara cepat menggunakan "*integral image*" .

3. Integral *image*

Integral Image digunakan untuk menentukan ada atau tidaknya dari ratusan fitur Haar pada sebuah gambar dan pada skala yang berbeda secara efisien. Pada umumnya, pengintegrasian tersebut berarti menambahkan *unit-unit* kecil secara bersamaan. Dalam hal ini *unit-unit* kecil tersebut adalah nilai-nilai piksel. Nilai integral untuk masing-masing piksel adalah jumlah dari semua piksel-piksel dari atas sampai bawah. Dimulai dari kiri atas sampai kanan bawah, keseluruhan gambar itu dapat dijumlahkan dengan beberapa operasi bilangan bulat per piksel.



Gambar 2.9 Integral *Image*

Seperti yang ditunjukkan oleh gambar 2.9 di atas setelah pengintegrasian, nilai pada lokasi piksel (x,y) berisi jumlah dari semua piksel di dalam daerah segiempat dari kiri atas sampai pada lokasi (x,y) atau daerah yang diarsir. Guna mendapatkan nilai rata-rata piksel pada area segiempat (daerah yang diarsir) ini dapat dilakukan hanya dengan membagi nilai pada (x,y) oleh area segiempat.

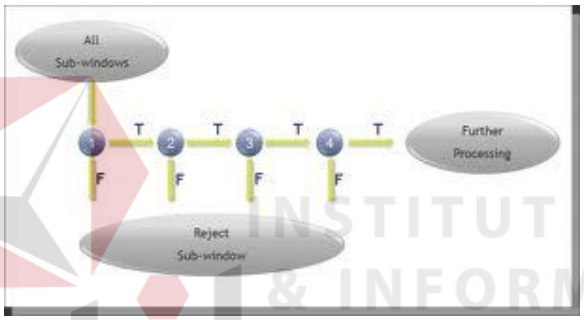
$$ii\,(x,y) = \sum_{x' \le x, y' \le y}^{n} (x', y') \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (2.25)$$

dimana (,)*ii x y* adalah *integral image* dan (,)*i x y* adalah *original image.* Guna mengetahui nilai piksel untuk beberapa segiempat yang lain, seperti segiempat D pada gambar 2.9**,** dapat dilakukan dengan cara menggabungkan jumlah piksel pada area segiempat A+B+C+D, dikurangi jumlah dalam segiempat A+B dan A+C, ditambah jumlah piksel di dalam A. Dengan, A+B+C+D adalah nilai dari integral *image* pada lokasi 4, A+B adalah nilai pada lokasi 2, A+C adalah nilai pada lokasi 3, dan A pada lokasi 1. Sehingga hasil dari D dapat dikomputasikan.

$$D = (A+B+C+D) – (A+B) – (A+C) + A \quad \cdots\cdots\cdots\cdots\cdots\cdots \quad (2.26)$$

4. Cascade classifier

Cascade classifier adalah sebuah rantai *stage classifier*, dimana setiap *stage classifier* digunakan untuk mendeteksi apakah didalam *image sub window* terdapat obyek yang diinginkan (*object of interest*). Stage classifier dibangun dengan menggunakan algoritma adaptive-boost (*AdaBoost*). Algoritma tersebut mengkombinasikan performance banyak weak classifier untuk menghasilkan *strong* classifier. *Weak* classifier dalam hal ini adalah nilai dari haar-like feature. Jenis *AdaBoost* yang digunakan adalah *Gentle AdaBoost*.



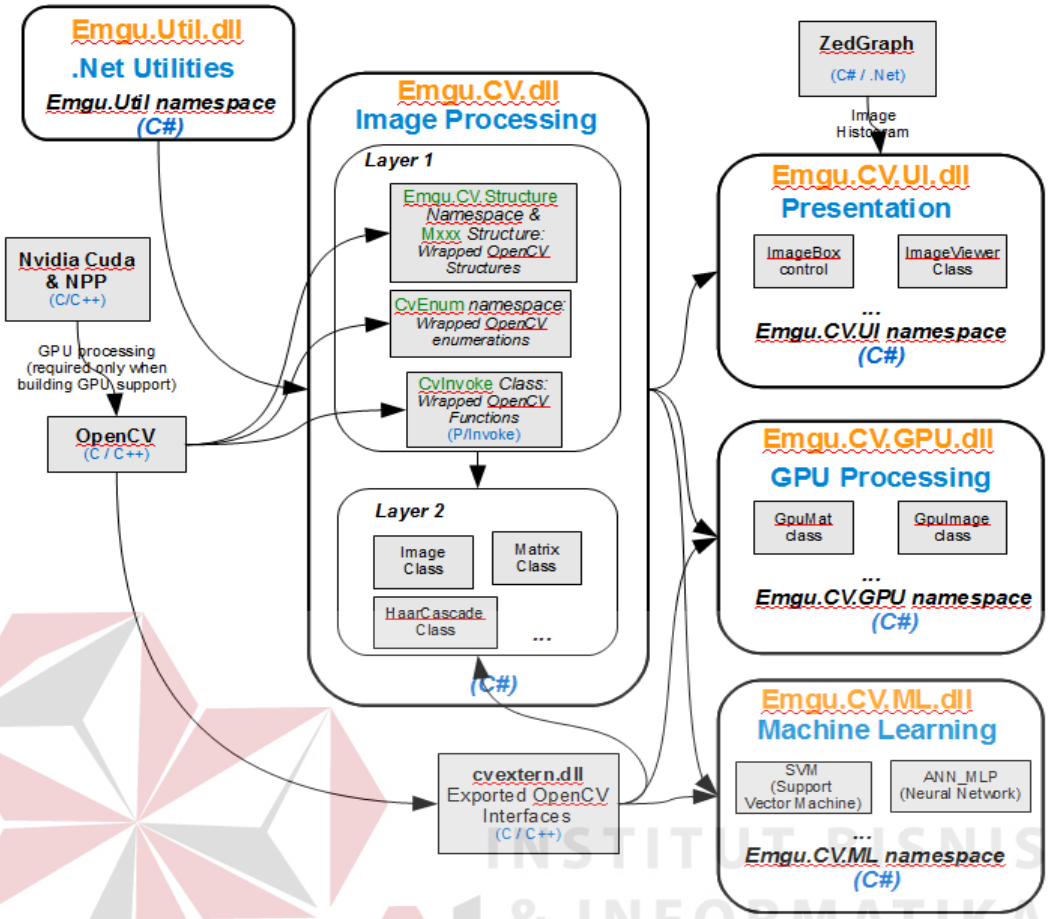
Gambar 2.10 Model *classifier* secara *cascade*

(Wahyu Setyo Pambudi, 2012)

2.12. *EmguCV*

EmguCV merupakan sebuah *library computer vision* untuk *platform* .NET (C#, VB.NET, *Ironphyton,* dan lain sebagainya). EmguCV adalah sebuah *cross platform* yang berfungsi untuk menjembatani antara OpenCV dengan semua bahasa yang kompatibel dengan .NET . Jadi semua fungsi yang dimiliki oleh OpenCV akan bisa digunakan juga oleh EmguCV, hanya saja akan berbeda dalam cara pemanggilan fungsinya. Fitur yang dimiliki EmguCV antara lain :

1. Manipulasi data citra (alokasi, *copying, setting,convertion*).
2. Citra dan video I/O (file and camera input based, *image*/video file output).
3. Manipulasi matriks dan vektor beserta rumus aljabar linier (*products, solver,eigenvalues,* SVD).
4. Data struktur dinamis (list, queues, sets, trees, graphs).
5. Pemroses citra fundamental (filtering, edge detection, corner detection, sampling and interpolation, color convertion, morphological operations, histograms, *image* pyramids).
6. Analisis struktur (connected component, contour processing, distance transform, various moment, template matching, hough transform, polygonal approximation, line fitting, ellipse fitting, delaunay triangulation).
7. Kalibrasi kamera (calibration patterns, estimation fundamental matrix, estimation homography, stereo correspondence).
8. Analisis gerakan (optical flow, segmentation, tracking).
9. Pengenalan obyek (*eigen methods*,HMM).
10. Graphical user interface (display *image*/video, keyboard handler, mouse handler, scroll bars).
11. Pelabelan citra (line, conic, polygon, text, drawing).



Gambar 2.11 *Library* dari EmguCV (EmguCV, 2012)

2.13. OpenRobotino *API*

OpenRobotinoAPI (*Application Programming Interface*) adalah *library* aplikasi *programming* yang dibuat khusus untuk Robotino yang diciptakan untuk mempermudah *user* dalam membuat program pada Robotino. *Library* ini memungkinkan akses penuh terhadap sensor dan *actors* pada Robotino. Komunikasi antara Robotino dengan PC melalui jaringan TCP dan UDP, dan semuanya sangat transparan, meskipun program yang berjalan sudah tertanam pada Robotino ataupun secara *remote*. (Robotino, 2010)